CSC2512
Algorithms for Solving Propositional Theories
CSC2512: Propositional Reasoning

**Instructor:** Fahiem Bacchus
- Office D.L. Pratt, Room 398B
- Office Hours:
  - Wednesday 3:30 to 4:30pm
  - Thursday 4:00 to 5:00 pm
  - by appointment (send email)
- [fbacchus@cs.toronto.edu](mailto:fbacchus@cs.toronto.edu) (please put CSC2512 in the subject)
- [www.cs.toronto.edu/~fbacchus](http://www.cs.toronto.edu/~fbacchus)

**Meetings:**
- Thursdays 1pm to 3pm. BA 2185

- Course Website: [www.cs.toronto.edu/~fbacchus/csc2512/](http://www.cs.toronto.edu/~fbacchus/csc2512/)
CSC2512: Propositional Reasoning

Evaluation:

– 30% Assignments.
  • Assignments will involve some programming. Typically you will have to use some software to model and solve a problem. There might also be some short answer questions.

– 20% Class participation. Paper summaries, discussion, etc.

– 50% Project
  • You can work individually or in teams of 2-3. The project will involve applying ideas from the course to solve some problem from your own area of research.
Overview:
• We will study various problems in propositional reasoning.
• These problems are complete for various complexity classes.
• The aim is to develop effective algorithms for solving these problems.
  – By effective we mean effective on a range of useful practical problems.

• Why?
  – If we can build effective solvers for a complete problem then any other problem within that problem class then then be solved by the “simple” device of encoding.
CSC2512: Propositional Reasoning

Practical Problem A that lies in NP

Encoding to SAT

Well engineered SAT Solver

Decoding

Solution to SAT problem

Solution to A without having to build a special purpose solver!
CSC2512: Propositional Reasoning

• Can this approach be successful?
  • Evidence with modern SAT solvers indicate that in fact this approach can sometimes offer significant performance improvements over developing problem specific software.

• In this course we will look at various complete problems and solvers for these problems.
• The most natural complete problems are propositional reasoning problems. (Logic plays a fundamental role of logic in computer science).
• By going beyond NP we look at other propositional reasoning problems.
CSC2512: Propositional Reasoning

Prerequisites:

• The course should be assessable to any CS or ECE graduate student.
  – Familiarity with programming, data structures and algorithms
  – Basics of propositional logic.
Types of Complete Problems we will examine (we will cover as many of these as we have time to):

• **Satisfiability.**
  – SAT, which is testing satisfiability over propositional theories
  – CSP (Constraint Satisfaction Problems), which is testing satisfiability over theories encoded as conjunctions of constraints (more general than CNF)
  – Many problems in scheduling, test generation, verification, etc. can be naturally encoded in SAT or in CSP

• **MAXSAT**
  – Optimization version of SAT. APX complete (so does not admit a polytime approximation scheme). And complete for $FP^{NP}$ the class of functions computable by a polynomial number of calls to an NP oracle.
  – Many important practical optimization problems can be encoded in MAXSAT.
CSC2512: Propositional Reasoning

Types of Complete Problems we will examine:

- **QBF (Quantified Boolean Formulas)**
  - PSPACE complete.
  - Offers compact encodings for many problems whose SAT encoding would be too large.

- **#SAT (if there is time/interest)**
  - Count the number of satisfying models.
  - Probabilistic reasoning in finite probability spaces can be reduced to #SAT.
CSC2512: Propositional Reasoning

Style of Course

- I will present lectures to provide needed background and then we will read some research papers to cover more recent topics.

- We will discuss these papers together. Your participation in these discussions will constitute the course participation component of your mark.
CSC2512: Satisfiability

Satisfiability: determine whether or not a satisfying truth assignment for a formula propositional F exists.

Various ways of forming propositional formulas

• Set of True/False propositional variables
• Set of logical connectives, typically including
  – Negation
  – Conjunction
  – Disjunction
• Sometimes,
  – Exclusive or
  – Implication
  – Not And (NAND), Not Or (NOR)
  – Etc.
CSC2512: Satisfiability

Satisfiability: determine whether or not a satisfying truth assignment for a formula \( F \) exists.

**Truth Assignments**

1. Truth assignment \( \pi \): map the propositional variables to True/False (0,1)
   
   \[ p_i \rightarrow \{0, 1\} \]

2. Extend to formulas:
   \[ \pi(\neg f) = 1 \text{ if } \pi(f) = 0 \]
   \[ = 0 \text{ if } \pi(f) = 1 \]
   
   \[ \pi(f_1 \land f_2) = 1 \text{ if both } \pi(f_1) = 1 \text{ and } \pi(f_2) = 1 \]
   \[ = 0 \text{ otherwise} \]
   
   \[ \pi(f_1 \lor f_2) = 1 \text{ if } \pi(f_1) = 1 \text{ or } \pi(f_2) = 1 \]
   \[ = 0 \text{ otherwise} \]
Satisfiability: Given a formula $F$ is there a truth assignment $\pi$ such that $\pi(F) = 1$?

With $n$ propositional variables we have $2^n$ possible truth assignments. Computationally hard in general to find a satisfying assignment from this large number of truth assignments.

Abbreviations: $f_1 \rightarrow f_2 = \neg f_1 \lor f_2$

$f_1 \equiv f_2 = (f_1 \rightarrow f_2) \land (f_2 \rightarrow f_1)$

$f_1 \oplus f_2 = (f_1 \land \neg f_2) \land (\neg f_1 \land f_2)$

$f_1 \oplus f_2 \oplus f_3 \oplus f_4 \oplus f_5$

ODD number of the $f_i$ are true.
Modern SAT solvers work with $F$ expressed in Conjunctive Normal Form (CNF)

**CNF**: a conjunction of clauses, each of which is a disjunction of literals, each of which is either a propositional variable or the negation of a propositional variable.

$$(p_1 \lor \neg p_2 \lor p_3) \land (p_2 \lor \neg p_5) \land (p_2 \lor \neg p_6) \land (p_4 \lor p_5) \land (\neg p_3)$$

We typically write this in abbreviated form:

$$(p_1, \neg p_2, p_3)(p_2, \neg p_5)(p_2, \neg p_6)(p_4, p_5)(\neg p_3)$$
1. A clause with clashing literals in it is true under any truth assignment. Such clauses are called tautological. Such clauses can be removed from the CNF
   • In general a formula is valid (tautological) if it is true under every truth assignment. It is unsatisfiable (the negation of a tautology) if it is false under every truth assignment. It is satisfiable if it is true under some truth assignment (if it is not a tautology and is satisfiable then its negation is also satisfiable).

2. Duplicate literals are irrelevant (the disjunction of two 0 or two 1 is still 0 or 1).

3. We say that a clause c subsumes another clause c’ if c is a subset of c’ when viewed as being a set of literals.
   • Any truth assignment that satisfies c must also satisfy c’
   • Subsumed clauses can be removed from the CNF without changing the set of models that make it true.
CSC2512: Satisfiability

Notes:

4. Each clause serves to eliminate some set of truth assignments (i.e., these truth assignments cannot be models of the CNF.
   - E.g., \((a, b, \neg c)\) eliminates all truth assignments \(\pi\) such that \(\pi(a) = 0, \pi(b) = 0,\) and \(\pi(c) = 1\)
   - Shorter clauses eliminate more truth assignments

5. By convention no truth assignment satisfies the empty clause ‘()’
   - (must satisfy at least one of the literals in the clause)

6. Validity is easy to detect.
   How?

7. Satisfiability is hard to detect (as hard as an arbitrary propositional formula).
CSC2512: Satisfiability

Disjunctive Normal Form, the dual of CNF.

**DNF:** a disjunction of terms, each of which is a conjunction of literals, each of which is either a propositional variable or the negation of a propositional variable.

\[(p_1 \land \neg p_2 \land p_3) \lor (p_2 \land \neg p_5) \lor (p_2 \land \neg p_6) \lor (p_4 \land p_5) \lor (\neg p_3)\]

One abbreviated form:

\[[p_1, \neg p_2, p_3] [p_2, \neg p_5] [p_2, \neg p_6] [p_4, p_5] [\neg p_3]\]
1. A term with clashing literals in it is false under any truth assignment. Such terms are **unsatisfiable** and can be removed from the DNF.

2. Duplicate literals are irrelevant (the conjunction of two 0 or two 1 is still 0 or 1).

3. A term \( t \) **subsumes** another term \( t' \) if \( t \) is a subset of \( t' \) when viewed as being a set of literals.
   - Only one of \( t \) or \( t' \) need be satisfied when satisfying the DNF, so we can remove \( t' \): any truth assignment satisfying \( t' \) must satisfy \( t \) so \( t' \) is redundant.
4. Each term serves to include some set of truth assignments (i.e., these truth assignments must be models of the DNF.
   - E.g., \([a, b, \neg c]\) includes all truth assignments \(\pi\) such that \(\pi(a) = 1\), \(\pi(b) = 1\), and \(\pi(c) = 0\).

5. By convention an empty term is valid (satisfied by all truth assignments).

6. Satisfiability is easy to detect
   How?
   What is an unsatisfiable DNF?

7. Validity is hard to detect (as hard as an arbitrary propositional formula).
Determining if there is a one anywhere (satisfiable) for $F$ becomes combinatorial as each clause makes a different set of truth assignments satisfying.
Determining if there is a zero (not valid) anywhere for $F$ becomes combinatorial as each term makes a different set of truth assignments satisfying...
Negation Formal Form

**NNF:** negations only apply to propositional variables (push all negations in using DeMorgan’s laws)

\[ \neg(f_1 \land f_2) = (\neg f_1 \lor \neg f_2) \]
\[ \neg(f_1 \lor f_2) = (\neg f_1 \land \neg f_2) \]
Converting to CNF.

If we have an arbitrary propositional formula $f$ how do we convert it to a CNF so that a SAT solver can be used?

- Could **multiply** out the formula to obtain a conjunction of disjunctions.

\[
\begin{align*}
(a \land b) \lor (c \land d) \\
(a \lor (c \land d)) \land (b \lor (c \land d)) \\
(a \lor c) \land (a \lor d) \land (b \lor c) \land (b \lor d)
\end{align*}
\]

- This leads to an CNF that can be exponentially larger than $F$ (but perhaps useful in limited contexts?)
Converting to CNF in time poly in the size (length) of $f$.

We introduce new variables (Tseitin 1970)

First convert to NNF (negation normal form)
CSC2512: Satisfiability

ToCNF(f)
  if f is a literal
    return (f, {})
  if f = f₁ ∧ f₂
    (g₁,{F₁}) = ToCNF(f₁)
    (g₂,{F₂}) = ToCNF(f₂)
    Let g be a new propositional variable
    return (g,{F₁} U {F₂} U {(-g, g₁), (-g, g₂), (-g₁, -g₂, g)})
  if f = f₁ ∨ f₂
    (g₁,{F₁}) = ToCNF(f₁)
    (g₂,{F₂}) = ToCNF(f₂)
    Let g be a new propositional variable
    Return(g,{F₁} U {F₂} U {(-g, g₁, g₂), (-g₁, g), (-g₂, g)})
CSC2512: Satisfiability

Notes:
1. The top level returns (g, F)
   - F is a set of clauses that capture the condition that g (a propositional variable) is equivalent to the original formula f
   - To test the satisfiability of f we add the unit clause (g) to F and test whether or not the CNF \{g U F\} is satisfiable.
   - How can we test validity?

2. All newly introduced variables are forced (must have a single value) under any assignment to the original variables.

3. F is always **satisfiable** and has $2^n$ satisfying models (where the original formula f has n variables).

Why?
CSC2512: Satisfiability

Example

\((a \land b) \lor (c \land d)\)

\(\text{ToCNF}((a \land b) \lor (c \land d))\)

\(\text{ToCNF}((a \land b))\)

\(= (g_1, \{(-g_1, a), (-g_1, b), (-a, -b, g_1)\})\)

\(\text{ToCNF}((c \land d))\)

\(= (g_2, \{(-g_2, c), (-g_2, d), (-c, -d, g_2)\})\)

\(= (g, \{(-g_1, a), (-g_1, b), (-a, -b, g_1), (-g_2, c), (-g_2, d), (-c, -d, g_2), (-g, g_1, g_2), (-g_1, g), (-g_2, g)\})\)

\(a = 1, b = 1, c = 1, d = 1 \implies g_1 = 1 \& g_2 = 1 \& g = 1\)

\(a = 1, b = 0, c = 1, d = 0 \implies g_1 = 0 \& g_2 = 0 \implies g = 0\)
Encodings: CNF is for most application not a natural language for expressing a problem. Various domains have different “standard” languages.

- Automated Planning: STRIPS or ADL actions specified with first-order variables
- Hardware: Circuits
- Software: Various specification languages (logics with extensions).

Specialized techniques have also been developed to encode problems expressed in these languages in CNF. The encoding can have a tremendous impact on how easy it is to solve the CNF.
CNF is used in modern SAT solvers mainly because there is a very simple reasoning rule that can be efficiently implemented.

**Definition: Resolution**
From two clauses \((A, x)\) and \((B, -x)\) (where \(A\) and \(B\) are sets of literals), we can produce the *resolvant* clause \((A,B)\) by a single *resolution* step. We write this as \(R[c,c'] = (A,B)\).

- Typically we require \(A\) and \(B\) not to contain conflicting literals (i.e., \(v\) and \(-v\) for any variable \(v\)). If they do then \((A, B)\) will be a tautological clause. Under this requirement, along with the restriction that there are no duplicate literals, the resolvant, if defined, is unique.
CSC2512: Satisfiability

Resolution is sound: Any truth assignment that satisfies c and c’ must satisfy R[c,c’] (if c and c’ are resolvable).
CSC2512: Satisfiability

Definition: Resolution Proof of a clause $c$ from a set of clauses $C$

A sequence of clauses $c_1, c_2, \ldots, c_n$ such that:

1. $c_n = c$ (the sequence ends in the proven clause $c$)
2. Each $c_i$ is either
   1. A member of the set of input clauses $C$
   2. or was derived by a resolution step from two prior clauses in the sequence $c_j$ and $c_k$ ($j, k < i$)

The sequence can also be represented as a DAG (directed acyclic graph).
CSC2512: Satisfiability

C = {(a, b) (-a, c) (-b, d) (-c, -d, e, f)}

There is a resolution proof of (a, b, e, f) from C:

(a, b), (-a, c), (c, b), (-b, d), (a, d), (-c, -d, e, f) (-c, a, e, f)
(b, a, e, f)
CSC2512: Satisfiability

Definition: **Resolution Refutation** of a set of clauses C is a resolution proof of the empty clause ‘(’ from C.

From soundness, any truth assignment satisfying C must satisfy the empty clause, but no truth assignment satisfies the empty clause ➔ Proves that C is unsatisfiable
Resolution is Complete:
If C is unsatisfiable there exists a resolution refutation of C.

Two computational difficulties
a) Finding a resolution refutation
b) Size of the refutation

Resolution is a relatively weak proof system. Well known families of CNFs whose shortest resolution proofs grow exponentially in size.
• Pigeon Hole Principle PHP first such problem shown to require exponential sized resolution proofs
• Other proof systems are known to have short proofs of PHP
Two computational difficulties

a) Finding a resolution refutation
b) Size of the refutation

Even if a short proof exists finding it might be hard

1. Notion of automatizability from proof theory.
2. For general resolution we can find a proof (given that one exists) in time $2^{O(n \log n \cdot \log S)}$ where $n$ is the number of variables and $S$ is the size of the shortest proof.
Davis Putnam (DP) [1960s] gave a procedure for determining satisfiability of a CNF formula. The procedure employs resolution in a systematic way so that if a resolution refutation is not found, none exists.

Ordered Resolution to test the set of clauses C:
1. Pick an ordering of the variables i[1], i[2], … i[n]
2. For j = 1 to n
   1. Let p = i[j] (the j’th variable)
   2. Let X = {all clauses of C that contain p}
      Y = {all clauses of C that contain –p}
   3. Apply resolution to all pairs of clauses from X and Y to obtain a set of new clauses R.
   5. If C contains {} return UNSAT
3. Return SAT
CSC2512: Satisfiability

Example:
(a, -b), (-a, b), (-b, c), (a, c), (a, -c), (-b, -c)

[a] X = (a, -b), (a,c), (a, -c)  Y = (-a, b),
   R = (b, c), (b, -c)

C’ = (-b, c), (-b, -c), (b, c), (b, -c)

[b] X = (b,c) (b, -c)  Y = (-b, c) (-b, -c)
   R = (c) (-c)

C’ = (c) (-c)

[c] X = (c)  Y = (-c)
   R = ()
Example:
(a, -b), (-a, b), (-b, c), (a, c), (a, -c),

[a] X = (a, -b), (a, c), (a, -c)  Y = (-a, b),
   R = (b, c), (b, -c)

C’ = (-b, c), (b, c), (b, -c)

[b] X = (b, c) (b, -c)  Y = (-b, c)
   R = (c)

C’ = (c)

[c] X = (c)  Y = {}
   R = {}

C’ = {}
Correctness:
Let $C' = \text{the set of clauses computed at step 2.3:}$
\[ C - X - Y + R \]

Claim: $C$ is satisfiable if and only if $C'$ is.

Note: If this is true then when $j = n$ either $C'$ is the empty set of clauses or it contains the empty clause. (All variables have been removed). Thus we can immediately determine if $C'$ is satisfiable or not. Working back to the prior clause sets $C$, we see that this determines whether or not the original clause set is satisfiable.
Correctness:
Let \( C' \) = the set of clauses computed at step 2.3:
\[
C - X - Y + R
\]

Proof: At stage \( j \) say that \( C' \) is SAT and let \( \pi \) be a satisfying truth assignment. Then there exists a setting \( v \) for \( p = i[j] \) such that \( \pi + (p = v) \) satisfies \( C \).

Find all clauses of \( C \) not satisfied by \( \pi \) (note \( \pi \) does not assign a value to \( p \))

Claim: all of these clauses contain \( p \) in only one polarity. If this is true then we simply extend \( \pi \) to make that polarity of \( p \) true, and thus satisfy all of the clauses of \( C \).
CSC2512: Satisfiability

Correctness:
Let $C'$ = the set of clauses computed at step 2.3:
$$C - X - Y + R$$

Proof:
Claim: all of the clauses of $C$ not satisfied by $\pi$ contain $p$ in only one polarity.

Say that this is not true, so there is $c_1 = (p, A)$ and $c_2 = (-p, B)$ unsatisfied by $\pi$, but then we have that $(A,B)$ is in $C'$, is satisfied by $\pi$, i.e., at least one literal from either $c_1$ or $c_2$ is made true, thus one of $c_1$ or $c_2$ must be satisfied by $\pi$.

For $(A,B)$ to be in $C'$, $c_1$ and $c_2$ must be resolvable, i.e., not a tautology. Why must this be the case?
CSC2512: Satisfiability

Correctness:
Let $C' = \text{the set of clauses computed at step 2.3:} \quad C - X - Y + R$

Proof:
Finally if $C$ is satisfiable so is $C'$: $C'$ contains only the clauses of $C$ and resolvants of clauses of $C$. 
CSC2512: Satisfiability

Davis Putnam (DP)

Ordered Resolution to test the set of clauses C:
1. Pick an ordering of the variables i[1], i[2], … i[n]
2. For j = 1 to n
   1. Let p = i[j] (the j’th variable)
   2. Let X = {all clauses of C that contain i[j]}
      Y = {all clauses of C that contain –i[j]}
   3. Apply resolution to all pairs of clauses from X and Y to obtain a set of new clauses R.
   4. C = C – X – Y + R    //CORRECTNESS. New C is satisfiable if and only if old C is.
   5. If C contains {} return UNSAT //C is UNSAT
3. Return SAT    //At this stage C = the empty set of clauses    //⇒ a satisfiable set of clauses